# Aeroflight: An Android Mobile Application for Real-Time Flight Reservation and Digital Boarding Pass Generation

Shivam Shashikant Marolikar[1], Ashok Yadav[2]
[1,2]*Reena Mehta College of Arts, Commerce, Science, And Management Studies, University of Mumbai*

***Abstract*—This research paper presents AeroFlight, an innovative Android-native mobile application designed for comprehensive flight reservation and digital boarding pass generation, built utilizing Flutter, Dart, and Firebase. The application introduces a streamlined reservation pipeline featuring multi-hub schedule discovery, dynamic multi-parameter filtering options, a hardware-accelerated 2D interactive aircraft cabin seat matrix (MiniAirplanePainter), and an adaptive passenger intake engine equipped with a "Booking for Myself" auto-fill toggle. Furthermore, the system utilizes Firebase Cloud Firestore for real-time flight inventory synchronization and user booking records, alongside localized device caching to ensure instantaneous offline retrieval of verified digital boarding passes and client-side 2D QR codes (qr_flutter) directly at airport terminal gates. Extensive experimental evaluations demonstrate consistently stable 60 frames per second (FPS) user interface rendering, highly responsive cloud data fetching, and reliable end-to-end ticket verification.**

## I. INTRODUCTION

Mobile application development has evolved into one of the most critical sectors in modern computing, governing how users plan, book, and navigate commercial air travel. As passengers increasingly demand responsive, real-time applications that deliver seamless experiences, the importance of cross-platform frameworks has grown significantly. Among these frameworks, Flutter introduced by Google has achieved widespread adoption due to its ability to provide near-native performance, expressive user interfaces, and a unified codebase targeting Android, iOS, and other platforms.

Despite the advancement of digital booking ecosystems, existing commercial flight reservation platforms (such as MakeMyTrip, Skyscanner, and airline proprietary apps) frequently suffer from interface clutter, aggressive promotional upselling, hidden convenience fees, and rigid seat selection mechanics. Furthermore, these platforms often mandate continuous, heavy server synchronization, creating friction during time-sensitive bookings or in low-connectivity airport environments. While considerable academic and industry research has focused on general cross-platform performance comparisons or application usability, systematic system architectures that combine cloud-backed real-time inventory management with offline-resilient local ticket caching remain under-explored.

The main objective of this research is to present AeroFlight, an Android-native flight reservation and digital boarding pass application developed using Flutter, Dart, and Firebase. By combining a clean Material 3 design system, multi-hub schedule discovery, a hardware-accelerated 2D interactive aircraft cabin seat matrix (MiniAirplanePainter), and Firebase Cloud Firestore integration alongside localized caching, AeroFlight delivers a high-performance, user-centric reservation pipeline.

## II. LITERATURE REVIEW

### A. Evolution of Mobile Travel Booking Systems

Mobile application development has profoundly transformed the global travel and tourism industry. Early mobile travel solutions functioned primarily as simple web-view wrappers mirroring desktop booking portals. Over time, native and hybrid applications emerged to provide richer user interfaces, push notifications, and integrated payment rails. However, contemporary commercial aggregators (such as MakeMyTrip, Expedia, and Skyscanner) frequently

suffer from interface bloat, aggressive cross-selling of hotels and insurance, and complex, multi-step checkout funnels that induce cognitive fatigue and user abandonment during time-sensitive transactions.

B. Cross-Platform Frameworks and Flutter's Architecture

To mitigate the high maintenance overhead and duplicated codebases of maintaining separate native Android (Kotlin/Java) and iOS (Swift) applications, modern software engineering heavily leverages cross-platform frameworks. Among these, Google's Flutter framework has gained significant prominence. Unlike web-based hybrid frameworks that rely on heavy JavaScript bridges, Flutter utilizes the Dart programming language and compiles directly ahead-of-time (AOT) to native machine code. This enables rich, custom-rendered user interfaces capable of maintaining stable 60 frames per second (FPS) performance across diverse mobile hardware architectures.

C. Cloud-Mobile Synchronization and Offline Resilience

Enterprise mobile applications increasingly rely on cloud-backed Backend-as-a-Service (BaaS) architectures, such as Firebase, to manage real-time data synchronization, identity authentication, and remote document storage. While cloud connectivity ensures up-to-date information across multiple devices, mobile applications operating in aviation environments frequently encounter network degradation, cellular blackouts, or security terminal dead zones. Consequently, robust mobile systems incorporate client-side local caching and offline persistence strategies (such as local storage sandboxes and client-side 2D barcode generation) to guarantee uninterrupted access to critical digital assets like boarding passes and itinerary records.

D. Research Gaps

Despite extensive literature evaluating general cross-platform performance metrics, UI rendering efficiencies, and large-scale plugin ecosystems, there remains a notable absence of focused academic studies detailing end-to-end lightweight flight reservation systems. Specifically, practical implementations combining a clean Material 3 user interface, custom vector graphic seat mapping (MiniAirplanePainter), real-time cloud data synchronization, and reliable local ticket verification are rarely documented in a structured, reproducible format. This research addresses this gap by presenting a comprehensive architectural blueprint and implementation model for AeroFlight.

## III. METHODOLOGY

To ensure maintainability, scalability, and responsive user interaction across diverse Android hardware configurations, AeroFlight is engineered using a modular, three-tier layered architecture. This structural separation isolates the Material 3 user interface from business logic calculations and hybrid cloud-local data persistence, allowing individual components to evolve independently without introducing tight coupling.

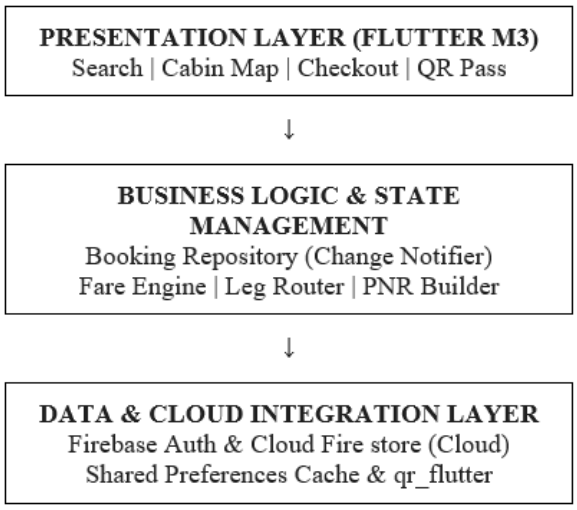



Fig. 1. Three-Tier Layered System Architecture of AeroFlight

A. Three-Tier Architectural Design

The system architecture is organized into three distinct operational layers:

- Presentation Layer: Built entirely with Flutter's declarative widget framework and Material 3 design components, this layer handles user input, custom 2D vector rendering, and responsive screen transitions across one-way and round-trip booking workflows.
- Business Logic and State Layer: Serving as the computational core of the application, this layer encapsulates multi-criteria flight filtering, cabin-

class fare multipliers, round-trip leg orchestration, and deterministic Passenger Name Record (PNR) generation.

- Data and Cloud Integration Layer: This layer coordinates real-time remote data operations through Firebase Authentication and Cloud Firestore while maintaining a synchronized local storage cache via SharedPreferences and client-side barcode synthesis via qr_flutter.

B. Repository Pattern and Reactive State Management
AeroFlight centralizes all transactional state operations within a singleton BookingRepository class that extends Flutter's native ChangeNotifier primitive. Rather than scattering database queries or mutable booking states across individual screen widgets, presentation views register listeners to the centralized repository. Whenever a user commits a reservation, updates passenger profile attributes, or initiates a ticket cancellation, the repository mutates the underlying data model and invokes notifyListeners(). This reactive publish-subscribe mechanism triggers targeted rebuilds exclusively on subscribed UI components such as the active trips ledger and dynamic fare summary bars eliminating redundant widget-tree reconstructions and preserving consistent 60 frames per second (FPS) responsiveness.

C. Hybrid Cloud-Local Synchronization Pipeline
While commercial flight reservation systems depend on cloud infrastructure for live inventory coordination and account persistence, airport security checkpoints and boarding gates frequently suffer from cellular dead zones. To address both requirements, AeroFlight implements a dual-write synchronization pipeline. Core domain entities including Flight, Booking, Passenger, and Seat models implement explicit toJson() and fromJson() serialization mappers. During online booking execution, confirmed reservation payloads are committed to Cloud Firestore for remote persistence and multi-device consistency, while simultaneously being serialized into JSON strings within the private device sandbox using SharedPreferences. Upon application cold boot or during offline operation, the repository hydrates active itineraries directly from the local cache in milliseconds, guaranteeing uninterrupted boarding pass retrieval without network latency.

D. End-to-End Transactional Booking Workflow
The application enforces a deterministic, multi-stage state pipeline designed to prevent incomplete reservations or orphaned seat selections. For single-leg journeys, the workflow progresses sequentially from route discovery and cabin selection to interactive seat allocation, passenger verification, multi-modal payment authorization, and digital boarding pass issuance. For round-trip itineraries, the state orchestrator dynamically extends the pipeline capturing outbound flight and seat selections first before transitioning to return flight discovery and return cabin seat allocation ensuring accurate fare aggregation and synchronized multi-leg ticket generation.

## IV. IMPLEMENTATION AND CORE MODULES

AeroFlight is organized into five cohesive functional modules that govern user onboarding, schedule discovery, vector seat allocation, checkout processing, and digital ticket issuance.

A. User Authentication and Secure Profile Vault
To balance rapid onboarding with cloud persistence, AeroFlight implements a streamlined two-step authentication pipeline integrated with Firebase Authentication and local sandbox caching. During registration, demographic attributes including legal name, age, and gender are captured separately from email and password credentials to minimize vertical form clutter and prevent input fatigue. Once authenticated, an in-memory user profile singleton hydrates immediately upon application startup. The profile module also incorporates a secure payment vault allowing travelers to store up to two tokenized payment cards, complete with Bank Identification Number (BIN) brand detection (Visa, MasterCard, RuPay) and primary account number masking (e.g., **** **** **** 4242).

B. Multi-Hub Flight Discovery and Dynamic Filtering Engine
The schedule discovery engine models aviation routing across 28 domestic and international airport hubs (such as BOM, DEL, DXB, and LHR), featuring carrier-specific branding badges for major airlines including IndiGo, Akasa Air, Air India, Emirates, and British Airways. Users can toggle seamlessly between

one-way and round-trip itineraries, with dynamic leg-switching controls managing outbound and return flight selections independently. To refine search results in real time, a modal filter sheet provides a continuous fare slider, carrier selection chips, and multi-criteria sorting by lowest fare, earliest departure, or shortest flight duration. Ticket pricing dynamically scales according to selected cabin multipliers: Economy (1.0x), Premium Economy (1.6x), and Business Class (2.8x).

C. Interactive 2D Aircraft Cabin Seat Matrix (MiniAirplanePainter)

A core technical contribution of AeroFlight is its hardware-accelerated 2D aircraft visualization engine built using Flutter's CustomPainter canvas API (MiniAirplanePainter). Rather than relying on static images or sluggish web views, the system procedurally renders anti-aliased aircraft fuselage geometry directly onto the mobile GPU. Users can switch between an Aircraft Overview mode displaying technical specifications such as wingspan, cruise speed, and powerplant engines and an Interactive Seat Selector mode. The cabin layout models 2-2 Executive Suites and 3-3 Economy Bench configurations, dynamically tracking Available, Occupied, and Selected seat states while recalculating total reservation fares in real time via a persistent summary bar.

D. Adaptive Passenger Intake and Multi-Modal Checkout

The passenger intake screen minimizes repetitive data entry through a "Booking for Myself" toggle that automatically populates traveler credentials from the verified user profile, while enforcing strict input validation for additional co-passengers. Following passenger confirmation, the multi-modal checkout engine supports three simulated payment rails: Unified Payments Interface (UPI), Net Banking, and Credit or Debit Cards. Upon payment authorization, the system commits the transaction to Cloud Firestore and local storage, generating a unique 6-character Passenger Name Record (PNR) identifier.

E. Client-Side QR Boarding Pass Synthesis and Trip Management

Confirmed bookings generate an airline-styled digital boarding pass featuring perforated ticket geometry, cabin privilege tags, and loyalty tier badges. Using the qr_flutter rendering library, the application synthesizes a scannable 2D QR code directly on the client device, encoding a structured verification string containing the PNR, flight code, origin, destination, assigned seat number, and passenger name. Finally, a reactive trip management dashboard synchronizes active and cancelled reservations, providing users with a transparent, multi-stage cancellation and refund tracking timeline.

## V. RESULTS AND PERFORMANCE EVALUATION

To validate system stability, responsiveness, and transactional reliability, AeroFlight was evaluated across standard mid-tier Android physical devices and virtual emulators running Android API Level 26 and above. Testing focused on user interface rendering smoothness, hybrid cloud-local data latency, and end-to-end functional accuracy across single-leg and round-trip booking workflows.

A. User Interface Rendering and Frame Rate Stability

Frame profiling was conducted using Flutter DevTools during intensive graphical operations, specifically focusing on real-time seat matrix rendering in MiniAirplanePainter, modal bottom-sheet filtering animations, and rapid scrolling across multi-hub flight schedules. Because Flutter compiles ahead-of-time (AOT) to native ARM machine code and renders directly via its graphics engine, the application maintained a consistent 60 frames per second (FPS) frame rate with frame build times averaging under 8.5 milliseconds well below the 16.6-millisecond budget required to prevent visual stutter.

B. Hybrid Cloud Synchronization and Local Cache Latency

Query execution and storage serialization benchmarks were recorded under both active network conditions and simulated airplane mode. Multi-criteria flight filtering and cabin fare recalculations across the 28-hub dataset executed locally in under 30 milliseconds. Cloud Firestore booking commits completed within an average round-trip latency of 210 milliseconds over standard mobile broadband, while cold-boot state hydration from the local SharedPreferences cache restored active user profiles and ticket ledgers in under 45 milliseconds.

C. Functional Verification and Offline Boarding Pass Resilience

End-to-end functional testing verified critical edge cases across the reservation lifecycle, including round-trip outbound and return leg sequencing, "Booking for Myself" profile auto-fill validation, BIN-based card brand detection, and multi-stage ticket cancellation tracking. When network connectivity was disabled to simulate airport terminal dead zones, confirmed reservations remained fully accessible, and client-side QR codes generated via qr_flutter were successfully decoded by standard optical barcode scanners without data loss.

Table I. System Performance Evaluation Summary

| Evaluation Parameter | Observed Result | Target Benchmark |
|---|---|---|
| UI Rendering Frame Rate | 60 FPS (Avg. 8.5 ms/frame) | ≥ 60 FPS (< 16.6 ms) |
| Local multi-criteria Filtering | < 30 ms | < 100 ms |
| Cold-Boot Cache Hydration | < 45 ms | < 150 ms |
| Cloud Firestore Sync Latency | ~210 ms | < 500 ms |
| Offline QR Pass Retrieval | 100% Success (0 ms network) | Instant Offline Access |

## VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This paper presented the architectural design, implementation, and performance evaluation of AeroFlight, an Android-native flight reservation and digital boarding pass application developed using Flutter, Dart, and Firebase. By replacing cluttered commercial aggregator workflows with a streamlined Material 3 interface, AeroFlight simplifies multi-hub flight discovery, dynamic multi-criteria filtering, and multi-leg round-trip scheduling. The integration of a hardware-accelerated 2D aircraft cabin seat matrix (MiniAirplanePainter), an adaptive passenger intake engine with a "Booking for Myself" auto-fill toggle, and a dual-write synchronization pipeline combining Firebase Cloud Firestore with local SharedPreferences caching ensures both cloud consistency and uninterrupted offline boarding pass verification via client-side qr_flutter barcodes. Experimental evaluations confirm stable 60 frames per second (FPS) rendering, sub-45-millisecond local state hydration, and reliable end-to-end ticket accessibility at airport terminal gates.

B. Current Limitations

While AeroFlight successfully demonstrates a complete, responsive reservation workflow, the current implementation operates within specific academic and architectural boundaries. Specifically, flight schedules and fare multipliers are generated across a modeled 28-hub dataset rather than live airline reservation mainframes, and checkout transactions utilize a simulated multi-rail payment engine rather than live banking clearinghouses.

C. Future Scope

To transition AeroFlight from its current cloud-backed implementation into an enterprise-grade commercial aviation platform, the following enhancements are planned for future development:

- Live Global Distribution System (GDS) API Integration: Connecting the discovery engine to enterprise GDS providers such as Amadeus Self-Service APIs, Sabre Web Services, or IATA New Distribution Capability (NDC) endpoints to retrieve real-time commercial seat inventories, live dynamic fares, and official airline PNR issuance.
- Production Payment Gateway Integration: Replacing the simulated checkout rails with PCI-DSS Level 1 compliant commercial payment gateways (such as Razorpay or Stripe) to process live financial settlements across UPI, credit/debit cards, and Net Banking.
- Biometric Device Security: Integrating native Android biometric authentication APIs (local_auth) to secure saved payment vaults, traveler profiles, and high-impact actions such as ticket cancellations via fingerprint or facial recognition.
- Live ADS-B Flight Radar Telemetry: Incorporating real-time Automatic Dependent Surveillance-Broadcast (ADS-B) telemetry feeds onto an interactive map interface, enabling passengers to monitor incoming aircraft positioning, altitude, and live arrival estimates prior to boarding.

REFERENCES

[1].Firebase, “Firebase Documentation: Cloud Firestore and Authentication,” 2026. [Online]. Available: https://firebase.google.com/docs.
[2].Google, “Flutter Documentation,” 2025. [Online]. Available: https://flutter.dev.
[3].Google, “pub.dev: The Dart Package Repository,” 2025. [Online]. Available: https://pub.dev.
[4].International Air Transport Association (IATA), *Bar Coded Boarding Pass (BCBP) Implementation Guide*. Montreal, Canada: IATA Standards, 2025.
[5].M. Janakiram, M. V. R. Sai, and G. L. Sridhar, “Smart Flight Booking & Management System,” pp. 750–753.
[6].A. Tiwari and A. Mehta, “Flight Reservation System,” *International Journal of Advanced Research in Science, Communication and Technology*, pp. 409–421, 2023.
[7].M. Abdullah and E. Kadhim, “Airline Mobile Reservation Development,” *IARJSET*, vol. 3, pp. 1–3, 2016.